

XGC with Kokkos/Cabana: Plasma Physics on Summit and Beyond

A. Scheinberg^{1,4}, S. Ethier¹, G. Chen², S. Slattery³, R. Bird², E. D'Azevedo³, S.-H. Ku¹, CS Chang¹

In collaboration with ECP-CoPA

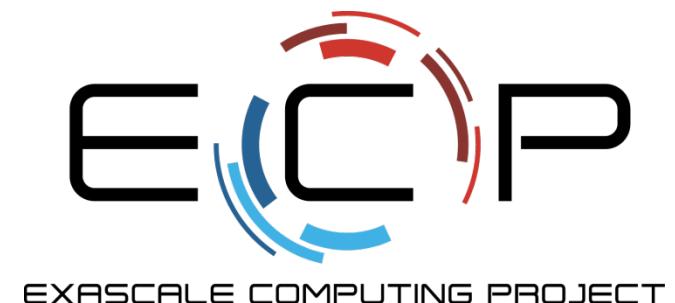
P3HPC 2020, Sept. 2

¹Princeton Plasma Physics Laboratory

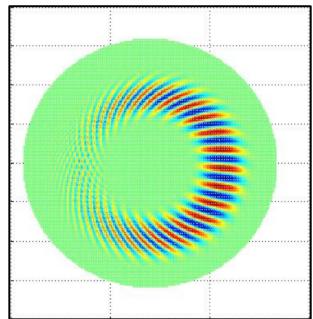
²Los Alamos National Laboratory

³Oak Ridge National Laboratory

⁴Jubilee Development

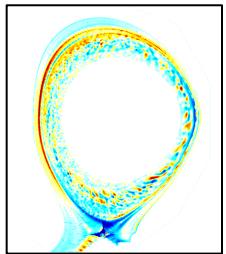


WDMApp Requires Exascale Computers and Beyond



Gigaflops

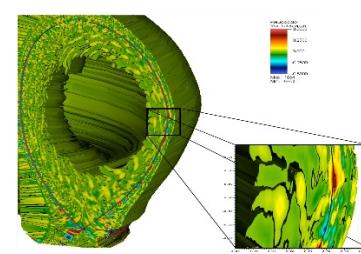
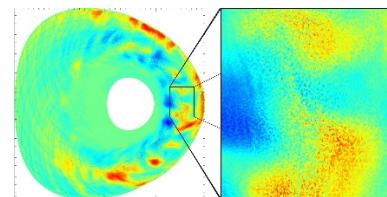
5-D electrostatic ion physics in simplified circular cylindrical geometry



Teraflops

Core: 5D ion-scale electromagnetic physics in torus

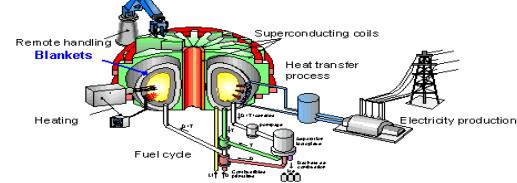
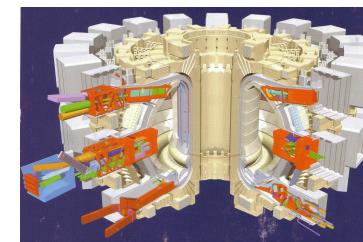
Edge: ion+neutral electrostatic physics in torus



Petaflops

Core: 5D ion scale Maxwellian ion + electron electromagnetic

Edge: non-Maxwellian plasma, electrostatic physics



Exaflops

Core-edge coupled 5D electromagnetic study of whole-device ITER, including ion-scale turbulence + local electron-scale turbulence, profile evolution, large-scale instability, plasma-material interaction, rf heating, and energetic particles

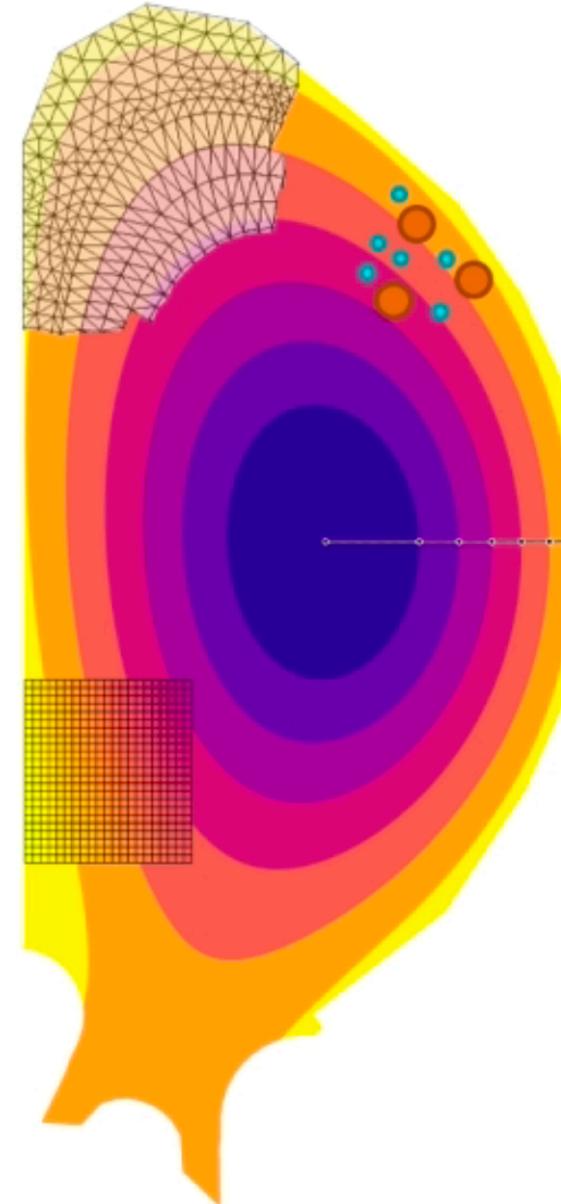
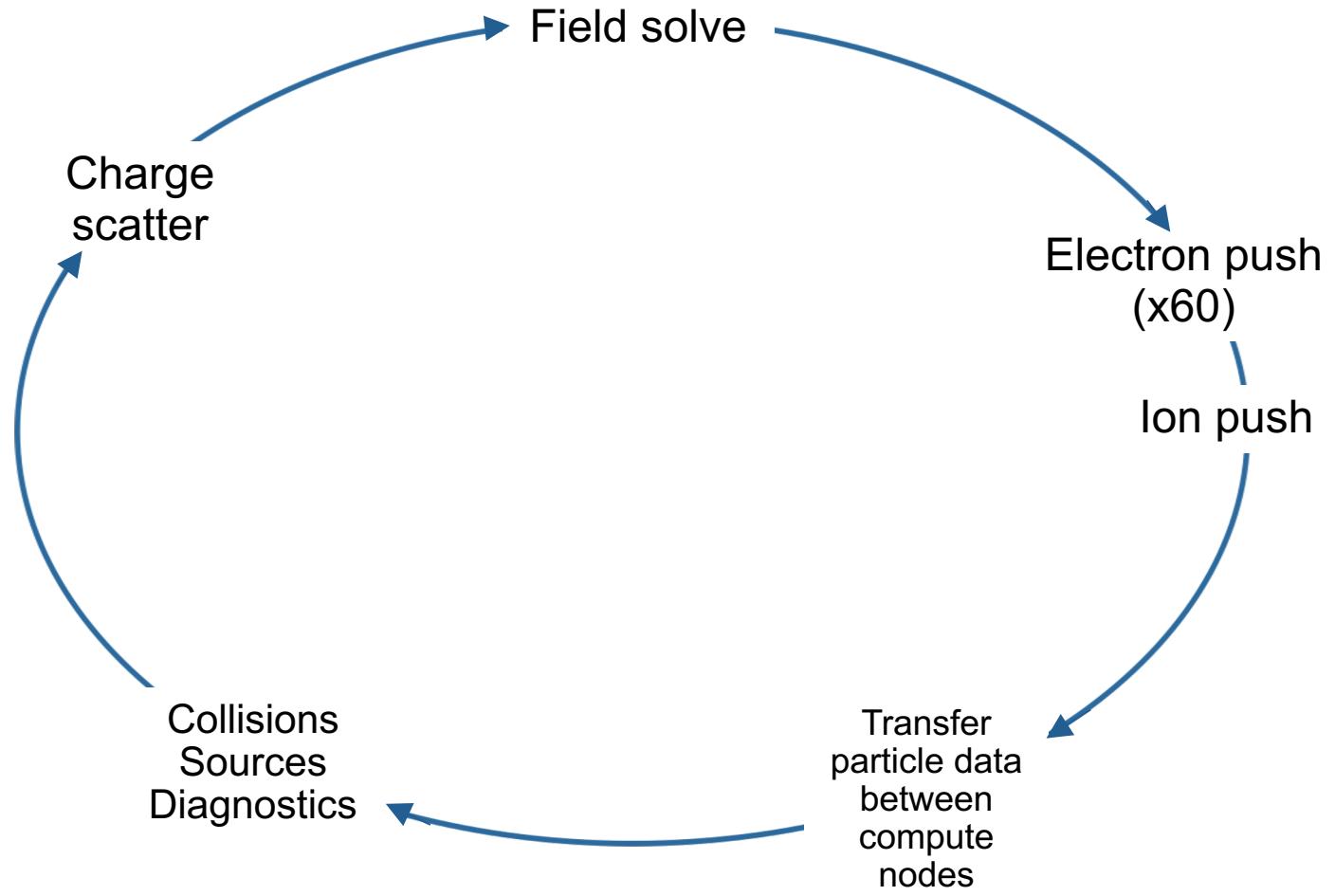
Beyond

A WDMApp that includes necessary engineering reactor components, and applicable to leading alternate concepts (including stellarators); and possibly 6D whole device modeling



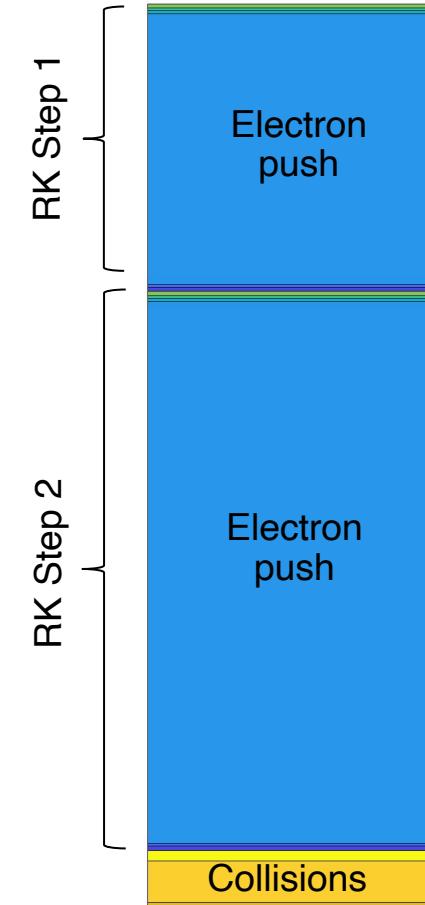
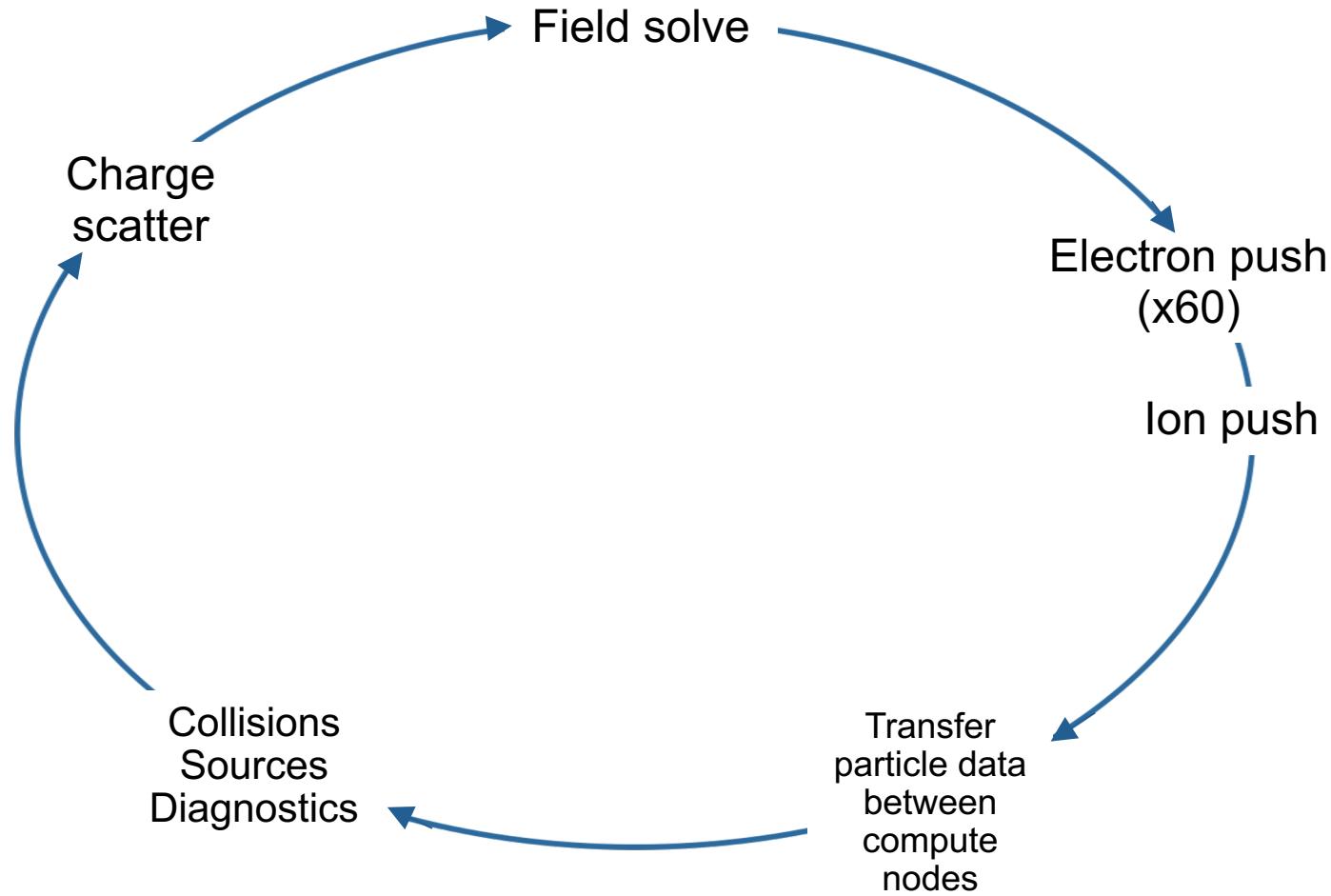
XGC outline

- Gyrokinetic (i.e. 5D) particle-in-cell code on an unstructured grid



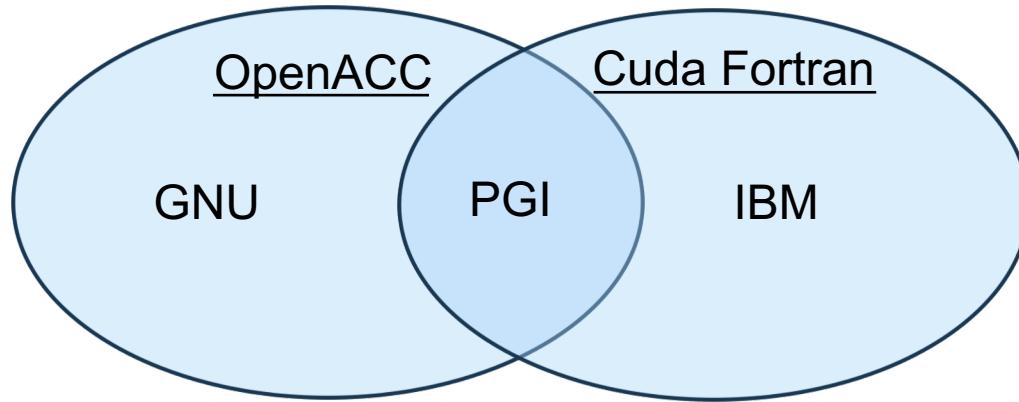
XGC outline

- Gyrokinetic (i.e. 5D) particle-in-cell code on an unstructured grid



Why adopt Cabana and Kokkos?

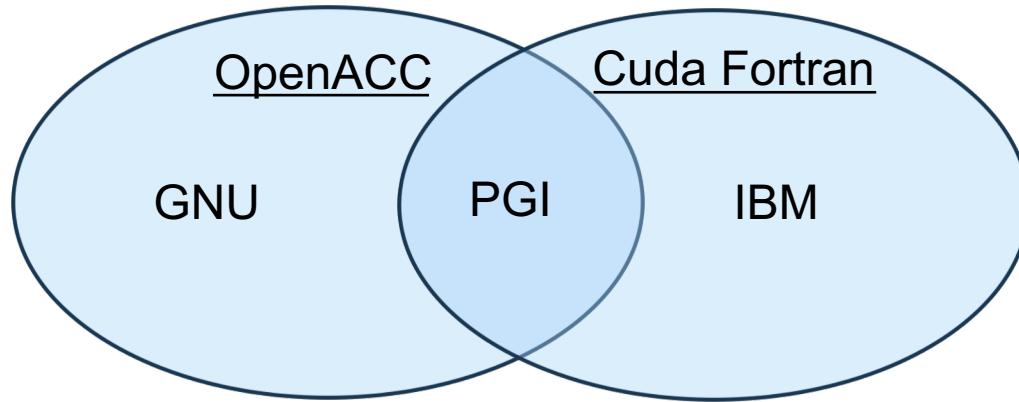
- Portability!
- Let Kokkos and Cabana handle **data management** and **kernel execution** for easy portability between architectures
- Reduce compiler dependencies (e.g. only PGI on Summit)



- Provide an easy/flexible framework for porting more kernels to GPU
- Avoid code duplication
 - 3 previous versions: original, vectorized, Cuda

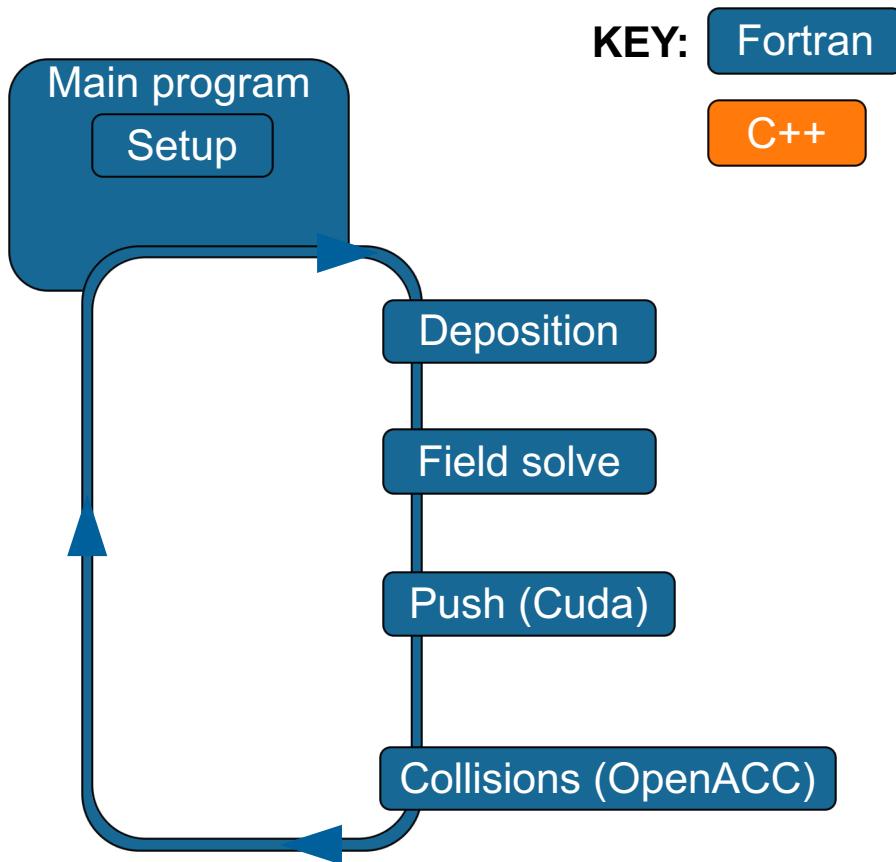
Why adopt Cabana and Kokkos?

- Portability!
- Let Kokkos and Cabana handle **data management** and **kernel execution** for easy portability between architectures
- Reduce compiler dependencies (e.g. only PGI on Summit)



- Provide an easy/flexible framework for porting more kernels to GPU
- Avoid code duplication
 - ~~3~~ previous versions: original, vectorized, Cuda, **Cabana**?
 - 4?

Old implementation of XGC

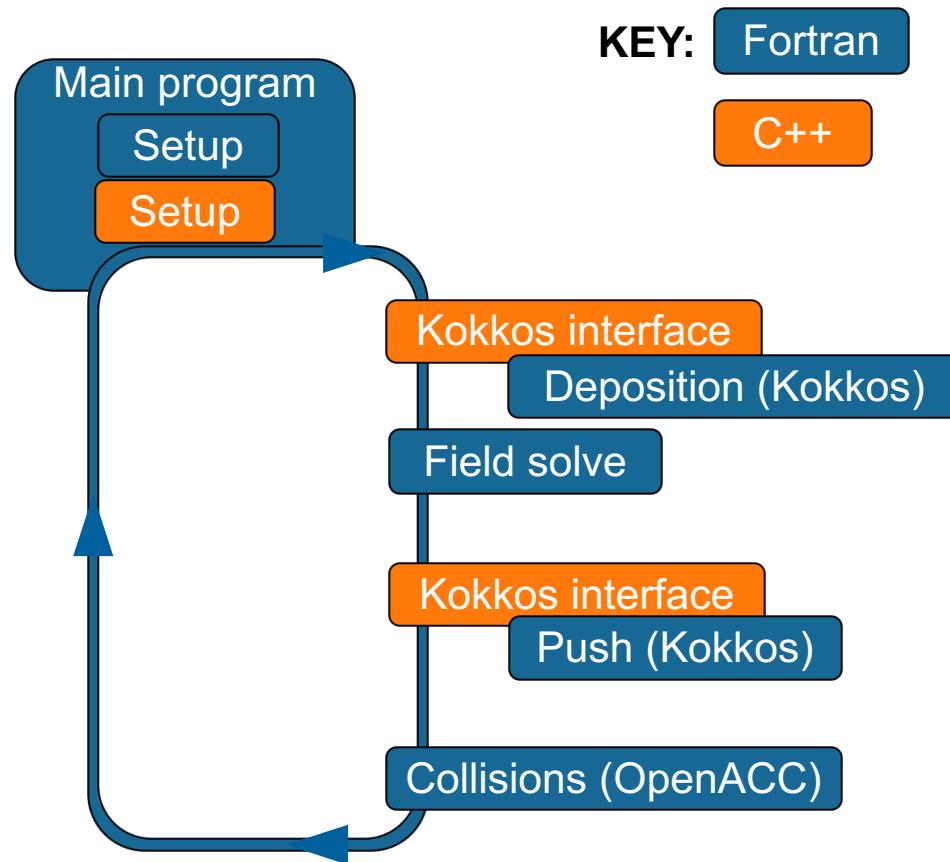


Old setup:

- All Fortran/Cuda Fortran

How does a Fortran code adopt a C++ programming model?

A Kokkos implementation of XGC



New setup:

- Keep Fortran main and kernels
- C++ interface
 - “Light touch:” Localized modification
 - Gradual implementation
- Unified, optimized code base

Data layout with Cabana

- Cabana (ECP-CoPA): a library for particle-based applications
 - Built on Kokkos
 - Provides AoSoA (array of structures of arrays) for versatile layout

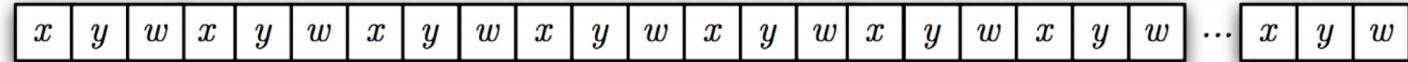
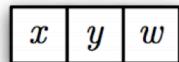
C++

```
// Define Cabana structure type          phase,    constants, global id
using ParticleDataTypes = Cabana::MemberDataTypes< double[6], double[3], int >;
```

Conceptual Layout

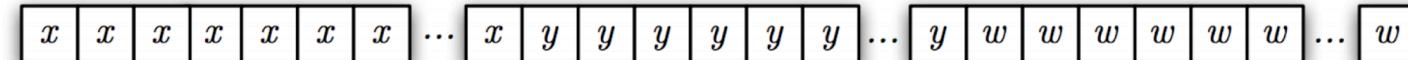
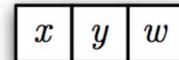
Physical Memory Layout

Array-of-Structs



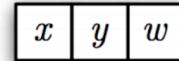
...

Struct-of-Arrays



...

Array-of-Structs-of-Arrays



SIMD Width

Executing the Kokkos parallel_for

- Kernel is called in a Kokkos parallel_for

C++

```
Kokkos::RangePolicy<ParticleList::array_size,ExecutionSpace> range_policy( 0, n_items ); // n_ptl on GPU, n_structs on CPU

// Execute parallel_for
Kokkos::parallel_for("my_operation", range_policy_vec, KOKKOS_LAMBDA( const int idx )
{
    push_f(p_loc+idx, idx);
});
```

- Must cast Cabana array into Fortran type for use in Fortran kernels

Fortran

```
module ptl_module
  use, intrinsic :: ISO_C_BINDING
  type, BIND(C) :: ptl_type
    real (C_DOUBLE) :: ph(vector_length,6)
    real (C_DOUBLE) :: ct(vector_length,3)
    integer (C_INT) :: gid(vector_length)
  end type ptl_type
end module
```

- Inner loops for vectorization on CPU

Fortran

```
subroutine push_f(particle_vec, i_vec) BIND(C,name='push_f')
  USE, INTRINSIC :: ISO_C_BINDING
  type(ptl_type) :: particle_vec
  integer(C_INT), value :: i_vec

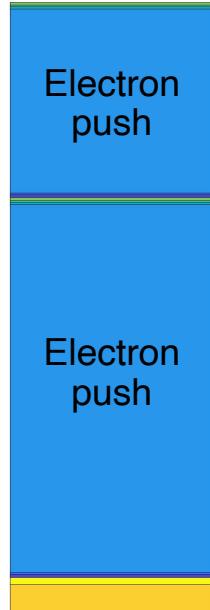
  do i=1, simd_size ! 32 on CPU, 1 on GPU
    ... ! Vectorizable loop that advances particle positions
  end do

end subroutine
```

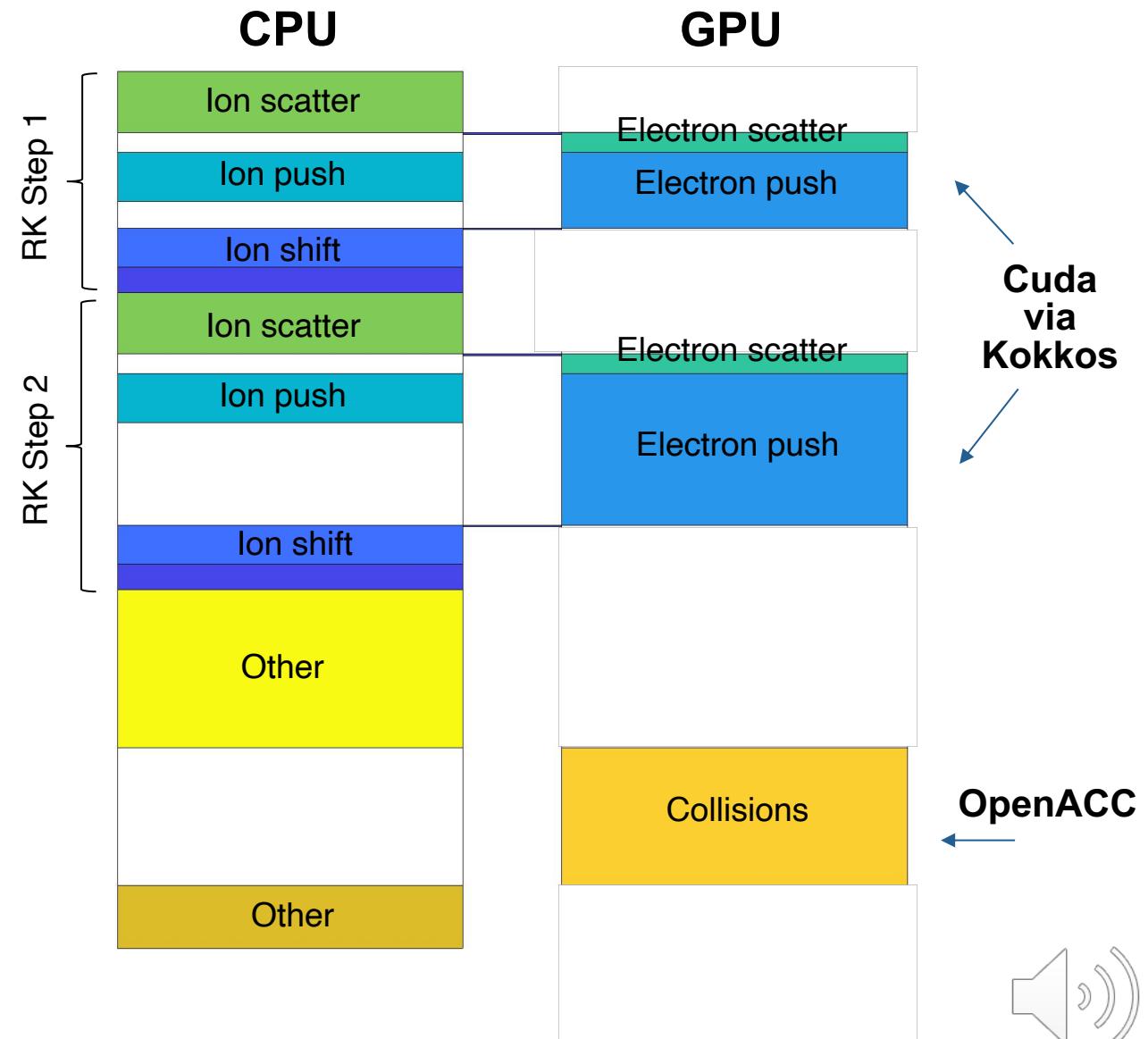
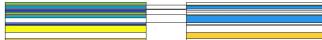
Timing on Summit (256 nodes)

- Overall speed-up: 15x CPU only
- CPU-GPU communication costs low
 - Actual electron transfer time shown
 - Favors simple approach to communication

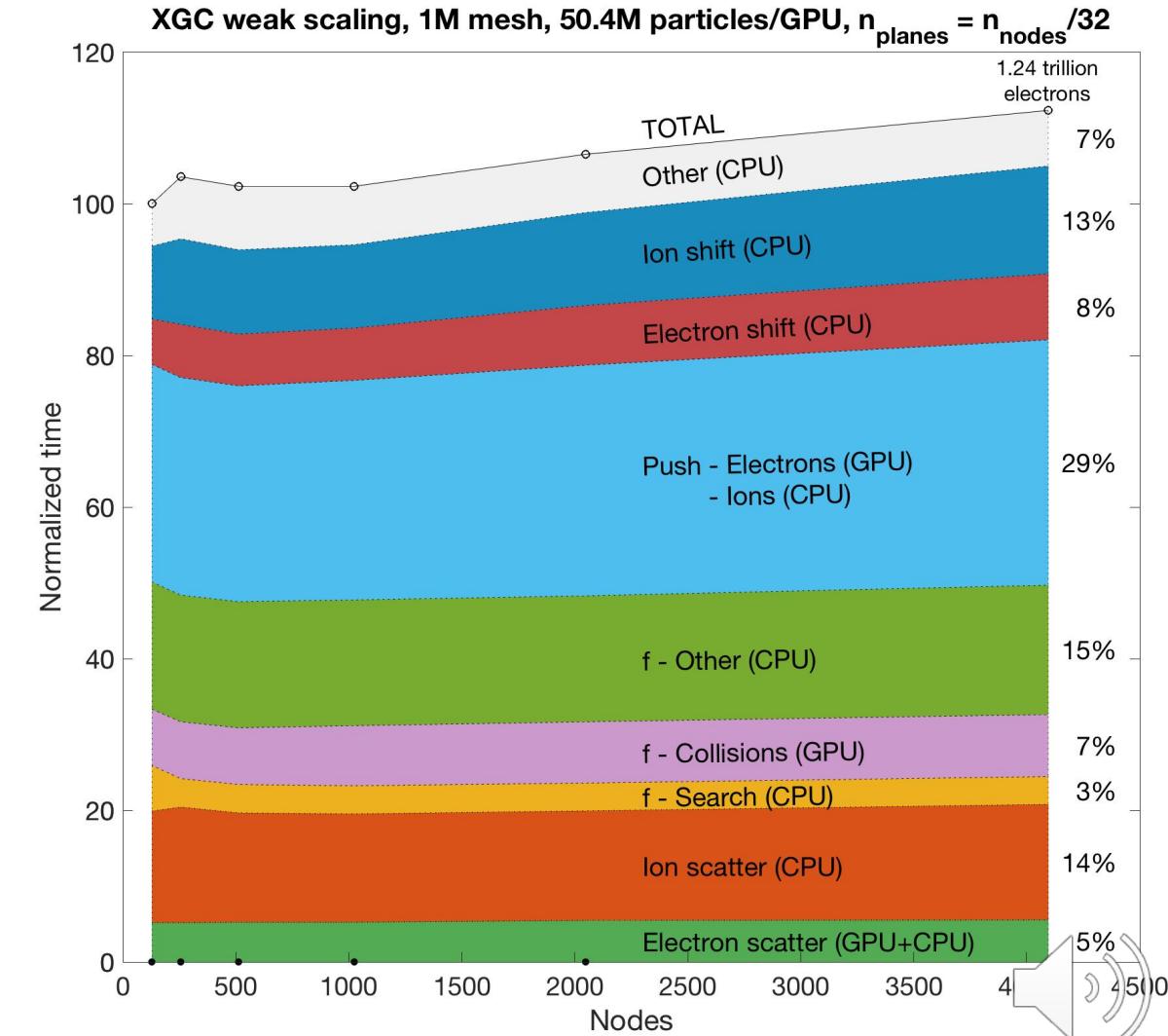
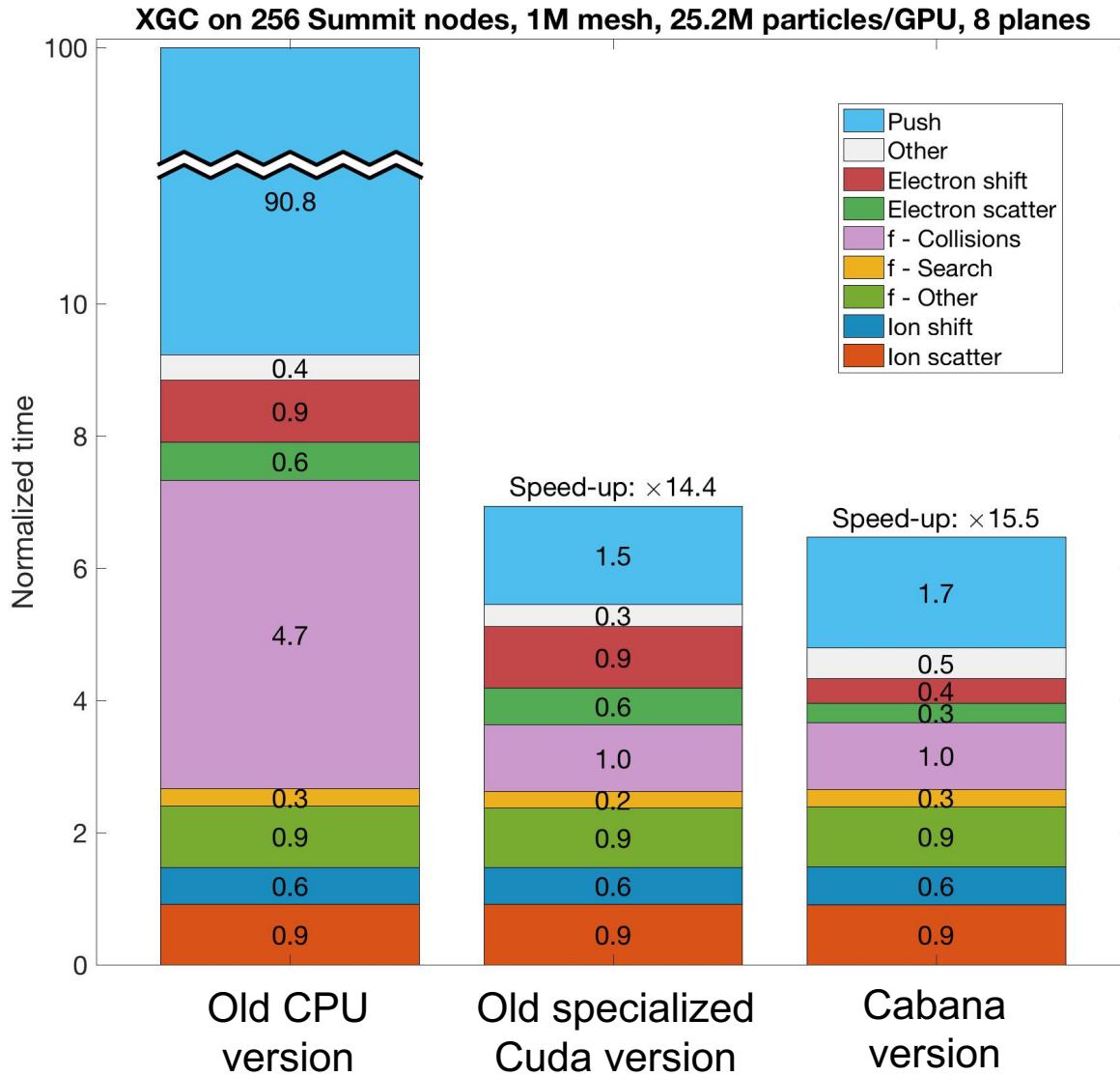
CPU-only



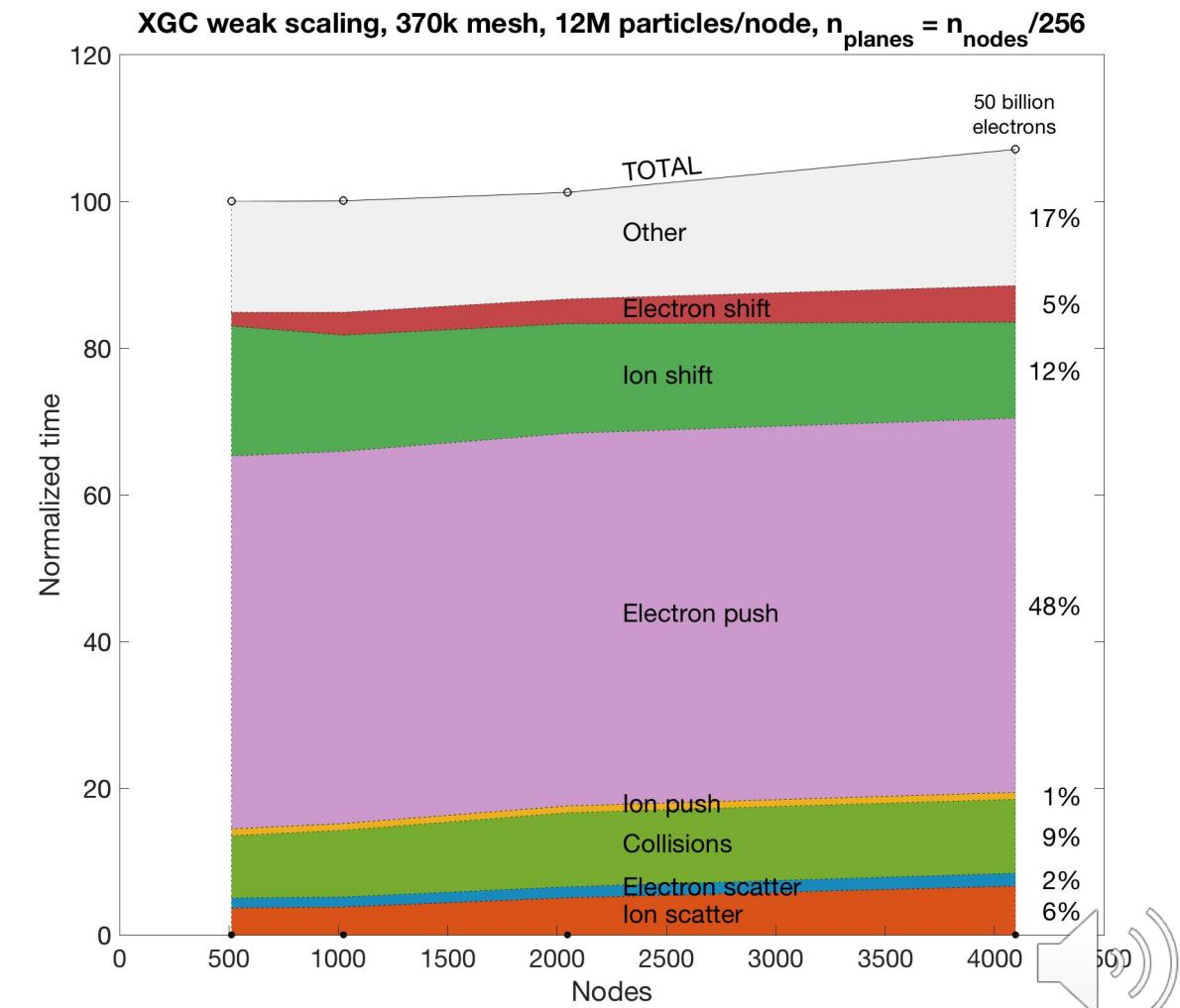
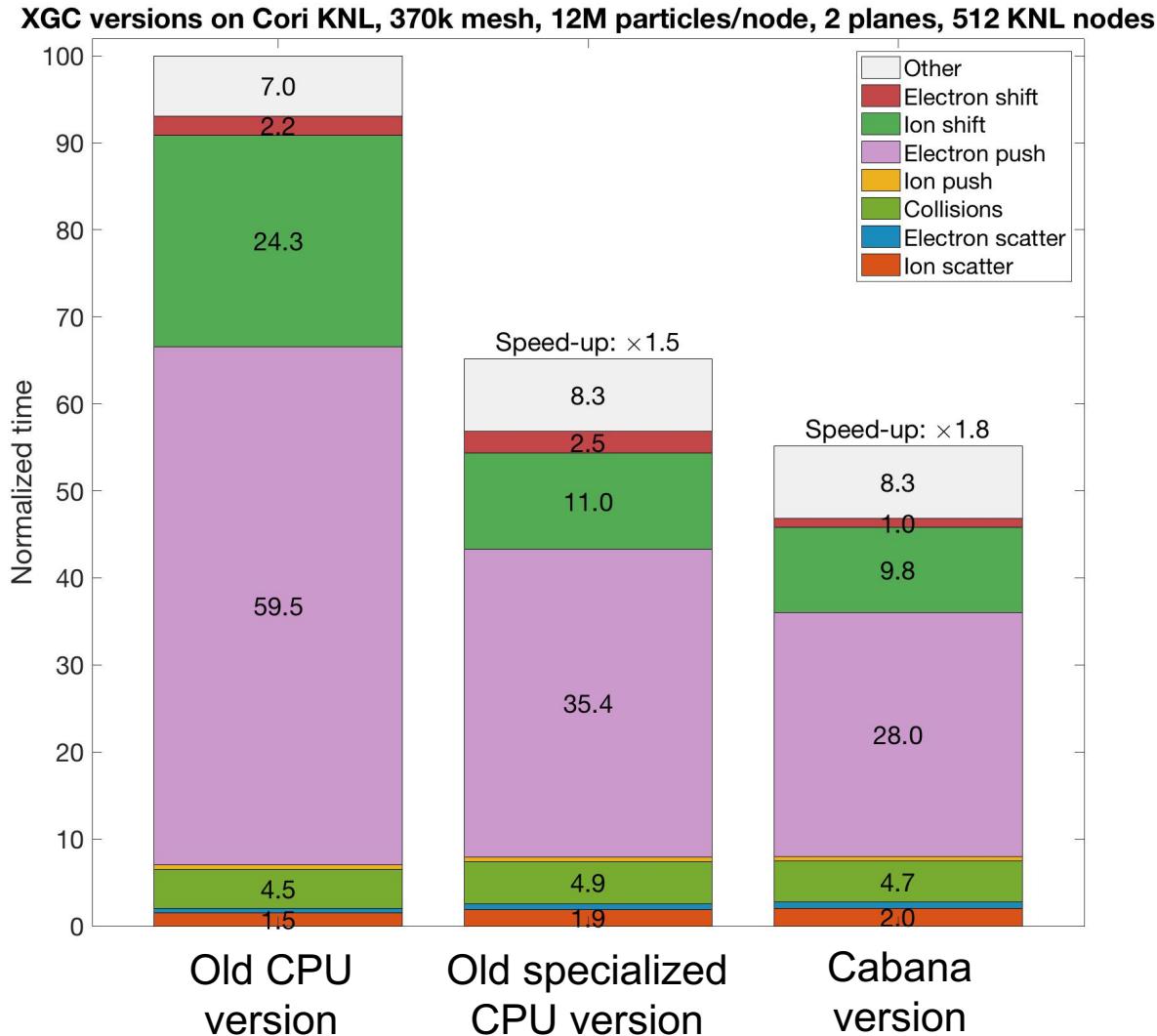
CPU+GPU



Summit performance comparison and scaling



Cori KNL performance comparison and scaling



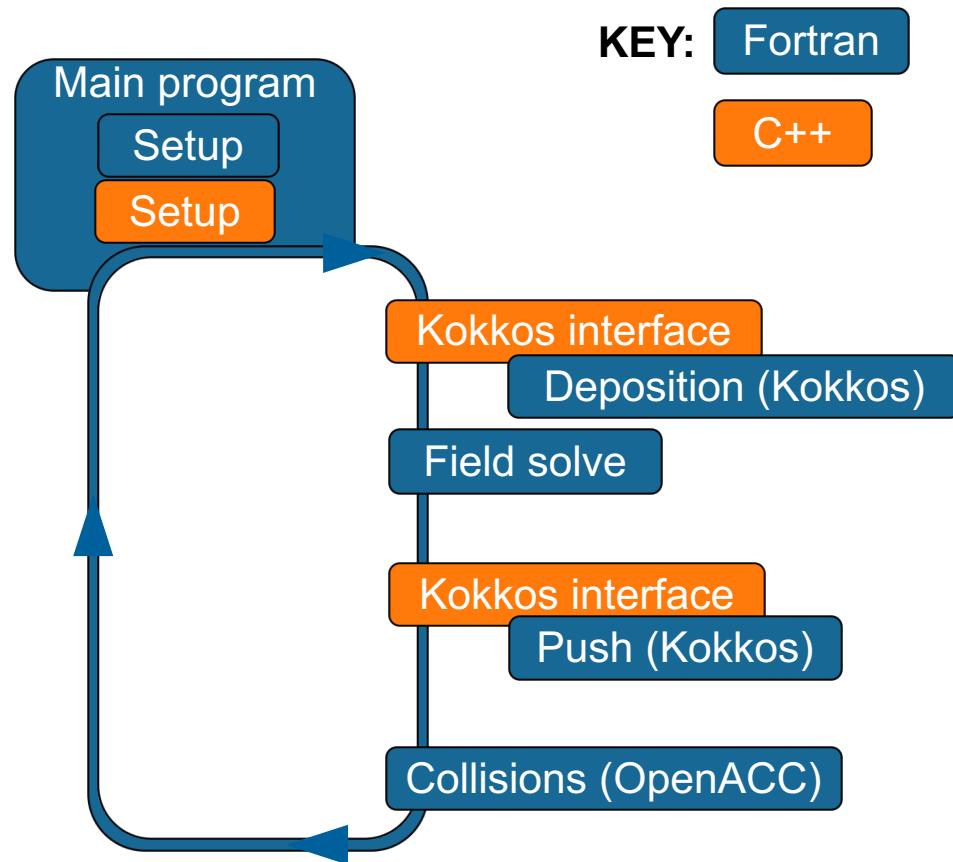
Transition to C++

- Diverse architectures coming up in the near future
 - Challenges lie ahead for portability

Supercomputer	Year	Petaflops	Architecture	Language
Summit	2019	200	Nvidia GPUs	Cuda
Perlmutter	2020	100	Nvidia GPUs	Cuda
Aurora	2021 (?)	1,000	Intel GPUs	SYCL
Frontier	2021	1,500	AMD GPUs	HIP
Fugaku	2021	1,000	ARM	Fortran/C++

- Support generally better for C++ than Fortran
- Easier use of Kokkos/Cabana if code is in C++

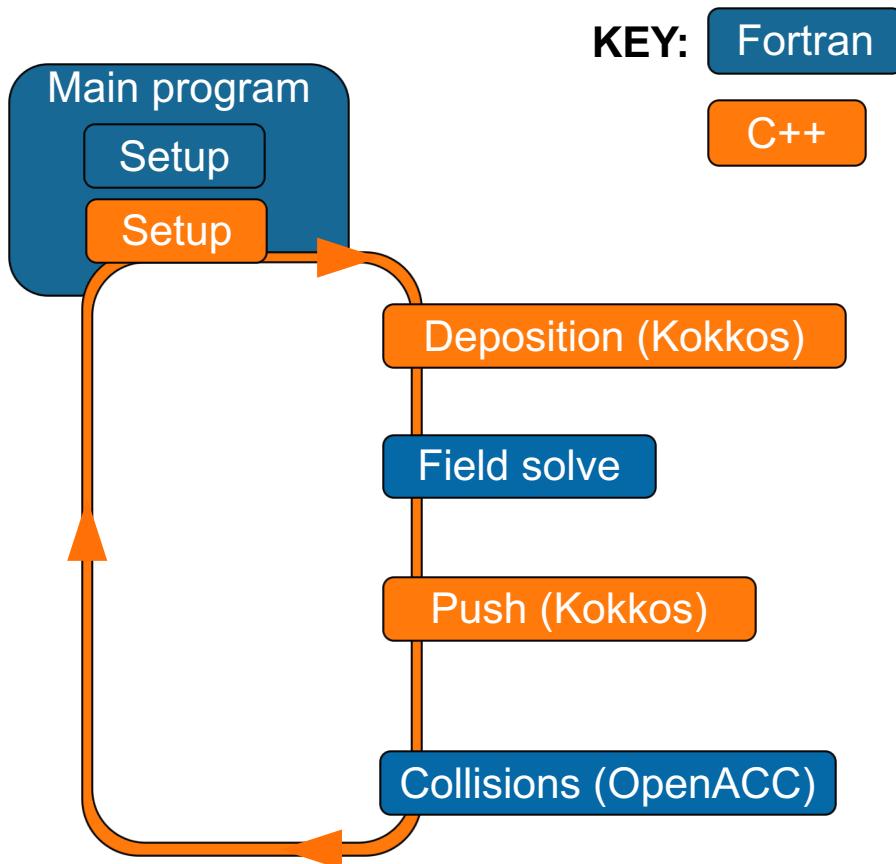
The Cabana Fortran implementation of XGC



The "Cabana Fortran" implementation

- Keep Fortran main and kernels
- C++ interface
 - “Light touch:” Localized modification
 - Gradual implementation
- Unified, optimized code base
- Downsides:
 - Inflexible macros
 - No HIP/SYCL support
 - Tedious data transfer

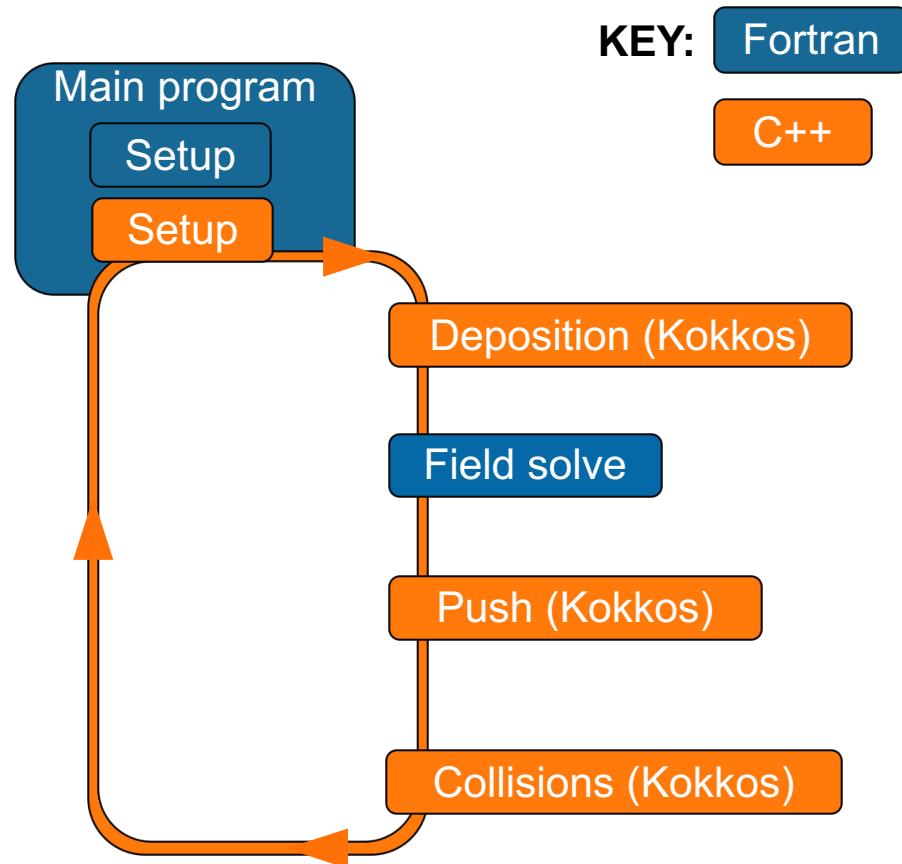
Kokkos C++ Implementation of XGC



Current setup:

- Transition to C++ continues
 - Main loop in C++ for easier memory management
- Integrated Kokkos/Cabana
- Arrays (field etc.) passed from Fortran, copied to Kokkos views
- No explicit cuda or OpenMP
- Ready for any architectures with Kokkos and OpenACC
 - In theory

Converting the collision kernel to Kokkos

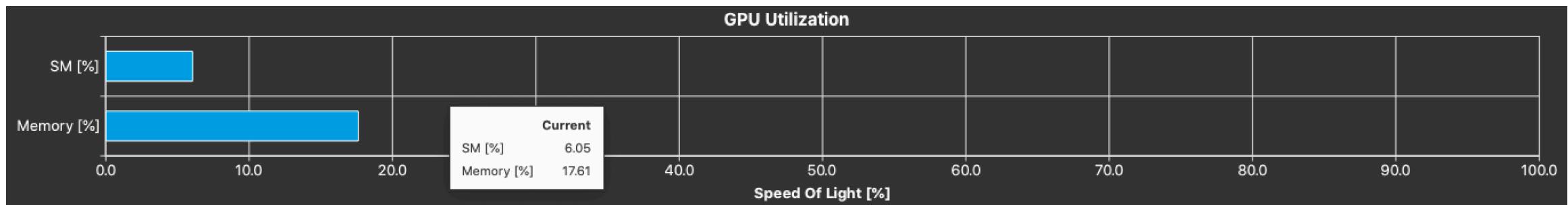
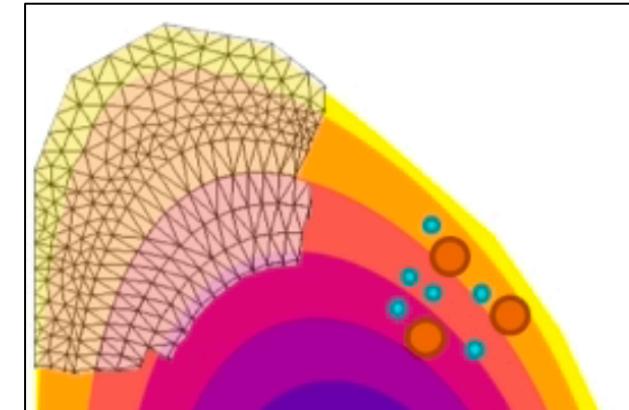


Motivation:

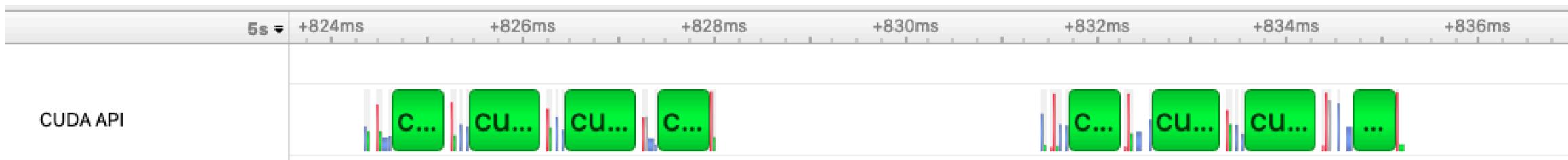
- Pitfalls of multiple programming models (Kokkos and OpenACC)
 - Memory management
 - Compiler compatibility
 - More opportunities for something to go wrong
- Converting to C++ anyway

Converting the collision kernel to Kokkos

- Problem: Collisions computed separately for each mesh node
 - ~5,000 mesh nodes per GPU
 - ~20 Kokkos kernels each
 - Kernels loop over ~1,000 elements -> GPUs underutilized

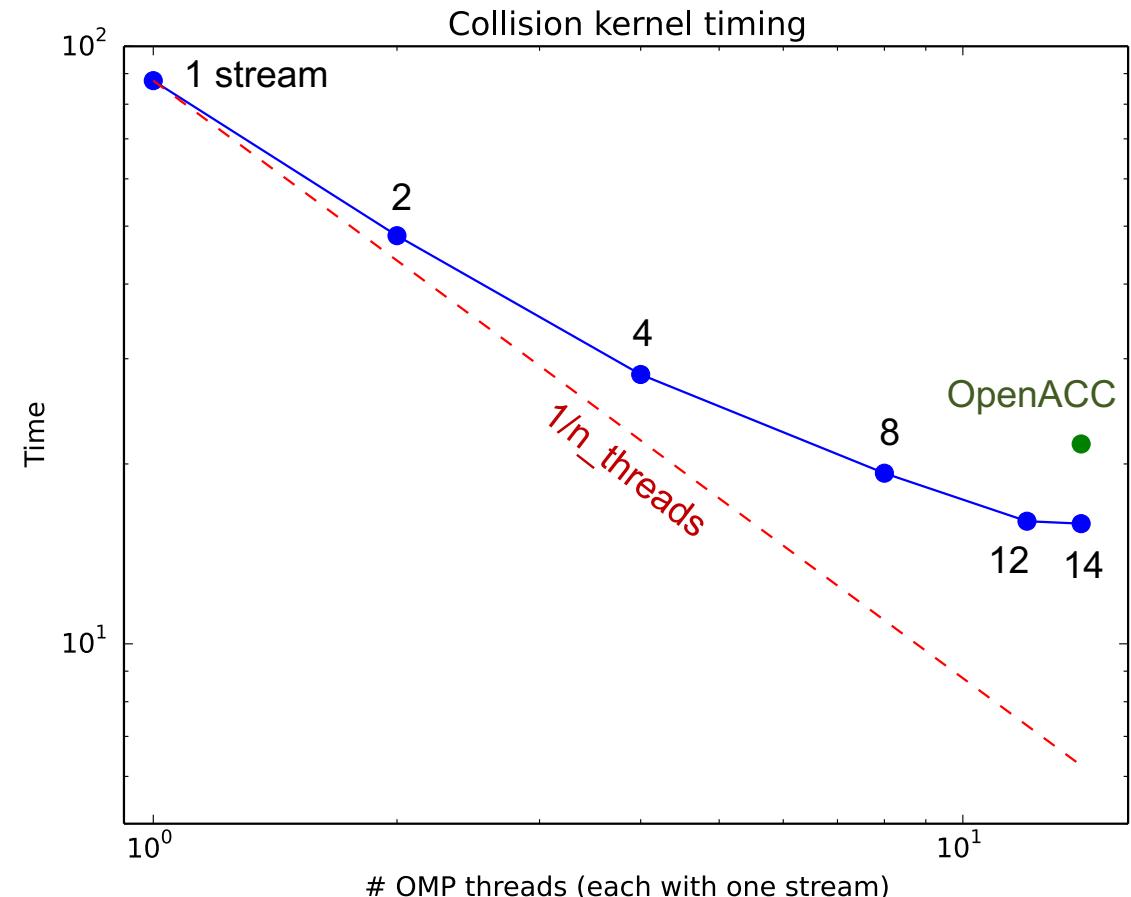


- Some calculations still on CPU (harder to port) -> More GPU idle time



Converting the collision kernel to Kokkos

- Approach: Multiple streams
 - Already done in our OpenACC implementation
 - Kokkos also supports Cuda streams
 - OpenMP parallel region, each OpenMP thread gets its own Cuda stream
- Result:
 - GPU usage much higher
 - 25% speed-up from OpenACC Fortran version
 - Still room for improvement (2-4x)?
- Downside: Possible portability challenges
 - Will multiple streams be a viable option for various Kokkos back-ends and architectures?



Summary

- XGC with Kokkos/Cabana is performing well on Summit and Cori KNL
- All major kernels offloaded to GPU with Kokkos
 - Electron push, collisions; also charge deposition, sorting
- More compiler flexibility (no longer tied to PGI on Summit)

Future challenges

- Moving more XGC kernels to Cabana framework
 - More optimization possible
- GPU-GPU communication
 - Potentially rely on Cabana for this
- Ensuring diverging developments can benefit
 - ECP-WDM projects (coupling with GENE, GEM, HPIC for whole-device modeling) and other science goals are on different branches